IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 13, 2020, accepted July 27, 2020, date of publication August 10, 2020, date of current version August 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3015346

TRACK-Plus: Optimizing Artificial Neural Networks for Hybrid Anomaly Detection in Data Streaming Systems

AHMAD ALNAFESSAH[1,2], (Member, IEEE), AND GIULIANO CASALE[1], (Member, IEEE)
[1]Department of Computing, Imperial College London, London SW7 2AZ, U.K.
[2]National Center for Artificial Intelligence and Big Data Technologies, King Abdulaziz City for Science and Technology (KACST), Riyadh 12354, Saudi Arabia

Corresponding author: Ahmad Alnafessah (a.alnafessah17@imperial.ac.uk)

This work was supported in part by the National Center for Artificial Intelligence and Big Data Technologies, King Abdulaziz City for Science and Technology (KACST), Saudi Arabia, and in part by the European Union's Horizon 2020 Research and Innovation Program, RADON, under Grant 825040.

ABSTRACT Software applications can feature intrinsic variability in their execution time due to interference from other applications or software contention from other users, which may lead to unexpectedly long running times and anomalous performance. There is thus a need for effective automated performance anomaly detection methods that can be used within production environments to avoid any late detection of unexpected degradations of service level. To address this challenge, we introduce TRACK-Plus a black-box training methodology for performance anomaly detection. The method uses an artificial neural networks-driven methodology and Bayesian Optimization to identify anomalous performance and are validated on Apache Spark Streaming. TRACK-Plus has been extensively validated using a real Apache Spark Streaming system and achieve a high F-score while simultaneously reducing training time by 80% compared to efficiently detect anomalies.

INDEX TERMS Apache Spark, artificial intelligence, big data, machine learning, neural networks, performance anomalies.

I. INTRODUCTION

In-memory processing technologies used for Big Data have been widely adopted in industry, in particular, Apache Spark has drawn particular attention because of its speed, generality, and ease of use. Here, we consider Apache Spark-based streaming workloads in which analytic operations are applied by means of resilient distributed datasets (RDDs). Our goal is to develop automated techniques to support performance anomaly detection. Although our focus is on a particular platform, elements of this approach may be exploited in the context of other stream processing systems.

Artificial Intelligence and machine learning algorithms are being increasingly used by researchers for performance anomaly identification and diagnosis [1]–[4]. Moreover, machine learning classification techniques are widely used to classify inputs based on their features into predefined classes to build a classifier that can predict the class of each item according to class labels. Popular classification techniques for performance anomaly detection include neural networks, support vector machines (SVMs) [5], and Bayesian networks [6].

The attention for anomaly detection is motivated by the fact that, with the growing complexity of Big Data and cloud systems, service-level management requires significantly higher levels of automation and attention [7]. An *anomaly* is defined as an abnormal behavior during the execution of a program. It could arise due to resource contention or service level disruptions, among several other factors. While some studies address the challenges of performance anomaly detection for batch processing [4], [8], [9], there is a demand for effective automated performance anomaly detection solutions specifically built for industrial-strength streaming systems, such as Apache Spark. This is because the platform does not natively report in log files either root causes of abnormal Spark tasks or information about when anomalous scenarios happen within the cluster [10]. Therefore, a practical solution is needed that can efficiently train a machine learning model

The associate editor coordinating the review of this manuscript and approving it for publication was Youqing Wang.

to identify performance anomalies within streaming workloads in production environments to produce such reports automatically.

This work is also motivated by the difficulty of carrying out anomaly detection within Big Data streaming systems, especially for time-varying workloads and critical applications. Apache Spark has more than 200 configurable parameters, and some parameters may depend on each other and affect the overall platform performance [11]. This large and complex configurable parameter space makes it difficult even for expert administrators to detect and classify anomalous performance within Spark Streaming clusters, as some performance level may simply depend on the chosen configuration. Therefore, the interaction between the performance of in-memory processing technologies and their configuration needs to be characterized in order to pinpoint and diagnose the root causes of anomalies, a classification task for which artificial intelligence methods are naturally well-suited.

This paper addresses the challenge of anomaly identification by investigating agile hybrid learning techniques for anomaly detection. We describe TRACK (neural neTwoRk Anomaly deteCtion in sparK) and TRACK-Plus, two methods to efficiently train a class of machine learning models for performance anomaly detection using a fixed number of experiments. TRACK revolves around using artificial neural networks with Bayesian Optimization (BO) to find the optimal training dataset size and configuration parameters to efficiently train the anomaly detection model to achieve high accuracy in a short period of time. TRACK-Plus is an automated fine-grained anomaly detection solution that adds to track a second Bayesian Optimization cycle for fine-tuning the hyperparameters of artificial neural network configuration. The objective is to accelerate the search process for optimizing neural network configurations and improving the performance of anomaly classification.

A validation based on several datasets from a real Apache Spark Streaming system is performed to demonstrate that the proposed methodology can efficiently identify performance anomalies, near-optimal configuration parameters, and a near-optimal training dataset size while reducing the number of experiments. Our results indicate that the reduction in experimental need can be up to 75% compared to naïve anomaly detection training. To the best of our knowledge, this paper is among the very first works that provide a comprehensive methodology for both performance anomaly classification and the efficient optimization of artificial neural networks to detect anomalies within streaming systems.

This paper extends a preliminary abstract in [12] by providing a comprehensive evaluation and classification model for three anomalous Spark Streaming workloads. In addition, the proposed methodology has also been enhanced by developing TRACK-Plus to simultaneously configure the artificial neural network used for anomaly detection. Our core contributions in this paper are as follows:

- Providing an updated discussion of existing anomaly detection techniques and algorithms that should be further researched by the community invested in this challenging problem space.
- Conducting a comparative analysis of four well-known anomaly detection techniques and algorithms to help system administrators in choosing the appropriate anomaly detection mechanisms for their in-memory Spark Streaming Big Data system.
- Addressing the challenge of anomaly identification and classification by investigating new hybrid learning techniques for anomaly detection in Spark Streaming Big Data systems.
- Presenting a comprehensive methodology to automate the search for the ideal dataset size with which to train the detection model and automate the tuning of neural networks hyperparameters to identify the most efficient network architecture and configuration.

The rest of the paper is organized as follows: A motivating example is comprehensively discussed in Section III, then the prerequisite background information about Apache Spark is presented in Sections IV. The proposed methodology of this work is presented in Section V, followed by a systematic evaluation in Section VI, and the results are discussed in Section VII. Related work is overviewed in Sections II. Finally, the VIII section presents a discussion and conclusions.

II. RELATED WORK

Performance anomaly detection techniques are important to optimize service levels in Big Data applications and large-scale distributed systems. Although the root cause of bottleneck and anomalous performance is often CPU congestion [13]–[15], Big data workloads are often also cache, memory, and network intensive, requiring advance techniques for their identification and mitigation.

Fulp *et al.* [5] use a machine learning approach to detect and predict the likelihood of service level disruptions using an SVM based on information from Linux system log files. They examine a dataset that contains over 24 months of actual file logs from a cluster with 1024 computing nodes. Their proposed solution achieves an acceptable level of classification performance of 73%. Fulp *et al.* [5], however, consider only one type of system failure —hard disk failure— without examining other common sources of systems failure, including CPU, cache, etc. Although SVM models are effective at making decisions from well-behaved feature vectors, they can be more expensive for modeling variations in large datasets and high-dimensional input features [16]–[18].

Qi *et al.* [8] propose a white-box model that uses classification and regression trees to analyze straggler root causes. The authors use raw metrics from Apache Spark logs and hardware sampling tools to train their model. The conventional decision tree algorithm has a drawback, however, which is the issue of overfitting. To avoid this issue, the authors use a special type of tree called a CART tree (*classification and*

regression tree) which offered some mitigation solutions. The solution includes a pruning technique (called CCP) when the tree growth is completed. The pruning process continues for several iterations and the classification performance metrics are checked for each node and its leaves [8]. Such a process is time-consuming, especially with intensive data streaming systems, so this study does not consider it. In addition, the presented work in [8] does not cover streaming processing workloads.

Lin *et al.* [19] propose an anomaly detection technique for infrastructure as a service (IaaS) cloud computing environment using local outlier factor (LoF) algorithm to detect anomalies by analyzing the reduced performance feature dataset. LOF is used to assign a score for each group of performance metrics to assess the system behavior, where the behavior is considered an anomalous if the score exceeds the predefined threshold. The authors validate their technique within a private cloud computing system that is built using OpenStack and Xen open-source software. Their result shows that the proposed technique outperforms principal components analysis (PCA).

Huang *et al.* [20] use an adaptive local outlier factor (LOF), a type of neighbor-based technique, for an anomaly detection scheme in cloud systems. They argue that their scheme could learn application behaviors in both training and detecting time. In addition, the scheme is adaptive to changes during the detection phase, which potentially significantly reduces the effort to collect the training dataset before the detection phase. The experimental results in [20] show that their scheme could detect performance anomalies with a low level of computational overhead. However, using the basic LOF requires considerable effort to collect enough datasets of normal behavior and requires intensive computations to calculate the distance scores of each instance during the test phase. According to [18], it is challenging to compute distance measurements for complex data, and such computations cannot identify some performance anomalies. Therefore, it is essential to keep in mind that the LOF needs to be adapted to be used with Spark Streaming for anomaly detection.

Table 1 further shows a summary of anomaly detection techniques used in the context of cloud and distributed computing systems. Further advancement in hybrid solutions holds great potential for anomaly identification systems [21], [22]. Some performance anomaly identification studies and surveys have been conducted in the literature for different purposes [14], [17], [23], [24]; however, there is still a shortage of studies that propose efficient automated anomaly detection, especially for in-memory Big Data stream processing technologies as we study in the next sections.

III. MOTIVATING EXAMPLE

In this section, we briefly illustrate the problem area and the benefits of Bayesian Optimization for anomaly detection. We have developed the customized benchmark *Network WordCountExp* for stream processing systems to generate our dataset for training purposes, more details are given in Section VI-B. Messages are sent to the data stream processing system with a fixed rate per second and number of lines per message. The inter-arrival time of messages is exponentially distributed. The Spark system is monitored at all times and we consider different levels of logging, ranging from measurements of Spark Streaming jobs to full recording of tasks execution logs, more details about Spark logging may be found in [41].

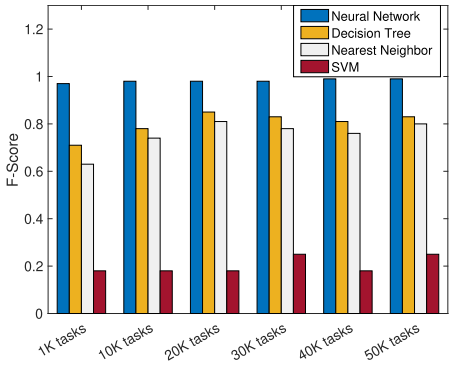
A detailed comparison is shown in Figure 1(a). The figure depicts the impact of Spark workload size, in terms of the number of tasks within the workload, on the neural networks model and comparison with other three well-known algorithms, namely nearest neighbor, decision tree, and support vector machine (SVM). The F-score metric is used to evaluate the accuracy of the neural network. Six training workload sizes with the same configurations are examined for sensitivity analysis, namely: 1000, 10000, 20000, 30000, 40000, and 50000 Spark Streaming tasks. From Figure 1(a), we see that the neural networks model outperforms all the other algorithms, achieving 98% F-score on average for the six different workload sizes. In comparison, the other methods feature a F-score on average 0.8% for decision tree, 0.75% for nearest neighbor, 0.2% for SVM. The computational complexity of neural networks depends on architecture of network, e.g. number of input features, number of layers, size of layer, etc. The complexity of the proposed neural networks is $O(m * N^{\frac{3}{2}})$, where m is number of iterations [42]. In terms of execution time, the neural networks, decision tree, nearest neighbor, and SVM took approximately 1 min, 2 min, 5 min, and 21 min, respectively. This example suggests that neural networks tend to be more effective than other AI/ML methods for anomaly detection within the Spark Streaming system, which motivates our interest to examine and train these models in streaming context.

We now examine an anomaly detection model that is trained using another neural network model. This model is trained with a single Spark Streaming workload configuration (with a *rate* of 2 *message/sec* and a *size* of 1000 *line/message*) by testing it against two unseen streaming workload configurations without injecting any anomalies. The first workload has a *rate* of 11 *message/sec* and a *size* of 1000 *line/message*, and the model achieves a 98% F-score. The second workload has a *rate* of 2 *message/sec* with a *size* of 5000 *line/message*, and the neural network model achieves a 98% F-score. These experiments demonstrate that the performance of the neural networks model is robust and unaffected by changes to streaming workload configurations when there are no anomalies.

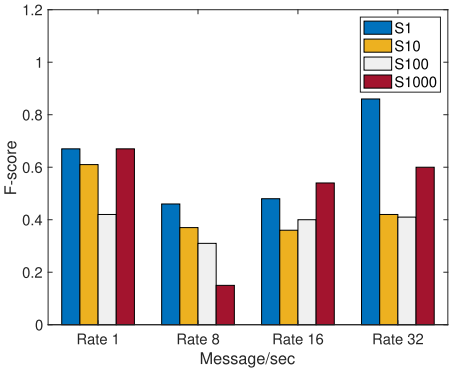
The same neural network is now trained on a single Spark Streaming workload configuration (with a *rate* of 2 *message/sec* and *size* of 1000 *line/message*) is typically used for some selected possible parameters of streaming workload configurations for *Sizes* 1, 10, 100, and 1000 *line/message* and *rates* of 1, 2, 4, 8, 16, and 32 *message/sec*, with artificial CPU anomalies injected. The F-score performance of the

TABLE 1. Summary of the state-of-the-art anomaly detection techniques.

Reference	Approach	Detection Technique	System/Environment
Magalhaes et al. (2010) [25]	Statistical	Correlation techniques and time-series alignment algorithms to spot the relationship between the transactions response time and the workload to pinpoint the occurrence of anomalies for dynamic workloads	Web-based and component-based applications applications
Zhang et al. (2007) [26]	Statistical	Regression-based model for estimating CPU demands by different client transactions. The result of the regression method is used to parameterize an analytic model of queues	Enterprise e-commerce system and TPC-W benchmark [27]
Kelly (2005) [28]	Statistical	Simple queueing-theoretic observations with standard optimization methods for performance anomaly detection	Distributed commercial systems that serve real customers
Yang et al. (2007) [29]	Statistical and Signal Processing	Extending the traditional window-based strategy by using signal-processing techniques to filter out recurring, background variations to determine which resource is the probable cause of an anomalous performance in a system	Three Grid environment applications: Cactus, GridFTP, and a Sweep3d
Lu et al. (2017) [10]	Statistical	Off-line approach to detect abnormal Spark tasks and analyze the root causes based on statistical spatial-temporal analysis. The mean and standard deviation of all tasks in each stage are used to get information about macro-awareness on the task's execution time	Private Apache Spark cluster and SparkBench [30]
Garraghan et al. (2016) [15]	Statistical	Empirical analysis for straggler detection and root-cause for batch processes using a combination of offline execution patterns modeling and online analytic agents for monitoring	Virtualized Cloud data centers
Bodik et al. (2010) [31]	Hybrid	Logistic regression to select a set of relevant metric to minimize both the prediction errors and quantile to summarize the values of each performance metric across all the application servers	Enterprise cloud computing and web applications
Chen et al. (2002) [32]	Hybrid	Using data mining and statistical techniques to correlate the normal and failure requests to pinpoint faults	PetStore, which is e-commerce environment based on the Java 2 Platform Enterprise Edition (J2EE) for Web and distributed systems.
Cohen et al. (2005) [33]	Machine learning	Pattern recognition, clustering, information retrieval, and Tree-Augmented Naive Bayes models (TAN) are used to characterize each metric and its contribution	Two distributed applications, one runs synthetic workloads in a private system and the other one run real customer services in a globally-distributed production environment
Fulp et al. (2008) [5]	Machine learning	Supervised detection and prediction of the hard disk failure using an SVM	Cluster with 1024 computing nodes
Pannu et al. (2012) [34]	Machine learning	Supervised one-class SVM and SVM for self-evolving anomaly identification and prediction	Utility cloud computing systems
Baek et al. (2017) [3]	Machine learning	Unsupervised labeling by clustering the training data set to create referential labels and supervised anomaly detection model that includes Naive Bayes, Adaboosting, SVM, and Random Forest	Enterprise and cloud computing systems
Ren et al (2018) [35]	Machine learning	Anomaly detection approach based on stage-task behaviors that related to the task execution status to classify normal and abnormal workloads according to the offline logistic regression model for each batch	Homogeneous Apache Spark Yarn Cluster for streaming workload with BigdataBench benchmark [36]
Lu et al (2018) [4]	Machine Learning	Anomaly detection using convolutional neural networks based model, which is implemented with different filters to automatically train model on the relationships among events	Big Data system logs using Hadoop distributed file system
Qi et al. (2017) [8]	Machine learning	White-box model for root-cause analysis of performance bottleneck and straggler based on CART decision tree	Private Apache Spark Cluster with HiBench benchmark to generate workloads
Yadwadkar et al. (2012) [37]	Machine Learning	Regression decision tree model periodically learns correlations between node level status and task execution time	Trace from Facebook Hadoop system and Berkeley EECS department's local Hadoop cluster (icluster)
Tan et al. (2011) [38]	Machine Learning	Semi-supervised one-class anomaly detection for streaming data using random half space trees	Open source datasets from KDD Cup 99 dataset [39]
Lin et al. (2011) [40]	Machine Learning	principal components analysis (PCA) for anomaly detection	Linux OS and high-speed network



(a) Comparison and sensitivity analysis of Spark workload sizes (the number of Spark tasks) on neural networks, decision tree, nearest neighbor, and SVM for CPU anomaly detection in Spark Streaming workloads.



(b) F-score for anomaly detection using neural network with *rates* 1, 8, 16, and 32 and *sizes* 1, 10 ,100, and 1000.

FIGURE 1. Motivation examples for TRACK. Figure 1(b) depicts a sensitivity analysis of four algorithms. Figure 1(a) shows the performance variance of neural networks with adjusted workload configuration parameters.

anomaly detection model dramatically decreases to a small figure between 0.1% and 3%. It is clear that the neural networks model fails to detect the CPU anomalies when the streaming workload configuration is changed. Therefore, the model requires additional training with more possible configuration parameters to detect anomalies. This baseline experiment demonstrates the critical need for a solution that would find the optimal dataset size and configuration parameters of a streaming workload for training the anomaly detection model within an in-memory Big Data system for generalization purposes.

Figure 1(b) shows some design factors and response variables (F-scores) for different streaming workload configurations where the proposed neural network model is trained using a single combination of configurations parameters (e.g., rate r and size s) and tested against other workloads stream configurations, which includes *rates* 1, 8, 16, and 32 *message/sec* and *sizes* 1, 10, 100, and 1000 *line/message*. As can be seen from Figure 1(b), it is not apparent which set of workload configurations would efficiently train the machine learning model to achieve the highest accuracy in a given time. The goal of this paper is to address the problem of joint optimization of neural network and experimental training.

IV. BACKGROUND INFORMATION

The following subsections briefly describe required background on Apache Spark Streaming, Bayesian Optimization, and neural networks.

A. APACHE SPARK STREAMING

Apache Spark stream processing has gained traction for a wide range of data processing applications in Big Data systems because of its ease of use, fault tolerance of stream data processing, and suitability of integration with other batch processing systems. Stream data can be ingested from many streaming sources to be processed and used by other systems [43]. Spark Streaming operates in a way that divides the entire received data stream into batches to be processed by the main Spark engine. The final data, i.e., the processed results, will consequently be segmented in batches. The input data stream can be fed from many different sources (Kafka, Flume, Twitter, etc.), and the stream data can be processed by advanced Spark libraries for machine learning and graph processing algorithms. The final output data from Spark Streaming can then be pushed out to databases or other systems [43].

Inside the Spark system, live stream data is fed to the Spark Streaming system. Spark Streaming divides the streaming workload into many batch workloads, which are then passed as inputs into the Spark core engine for data processing. In Spark Streaming, high-level basic abstractions are called discretized streams (*DStreams*) and are continuous streams of data. Each *DStream* is either an input data stream received from other streaming sources or it is the result of a processed data stream created from the input streams [43].

Internally, each *DStream* contains a sequence of Spark Resilient Distributed Datasets (RDDs), which are the main Spark Core data abstractions. RDDs cannot be changed and can be executed in parallel. In addition, RDD offers operations, including data transformation and actions, that can be used for Spark Streaming for data analysis. Each RDD in the *DStream* represents data for a specific time interval. Therefore, all operations are applied on *DStream* will be applied to the RDDs within the same *DStream* [43].

B. NEURAL NETWORK MODEL

The term *backpropagation* in neural networks comes from computing the error vector backward, starting from the last layer in the network [44]. Before backpropagation is initiated, other processes are done first. These processes include calculating the activation values of units and propagating them to the output units. Then the cost function is applied to compare the actual output error results y_P^o with the desired output values d_o. There is usually the signal error δ_o^P from each unit in the output layer. The goal of backpropagation is to reduce the amount of difference between the actual output and the desired output as much as possible [45]. This can be achieved by backward passes through every hidden layer in order to carry the error signal to all units in the neural networks and to recalculate the weights of connections in the hidden layers.

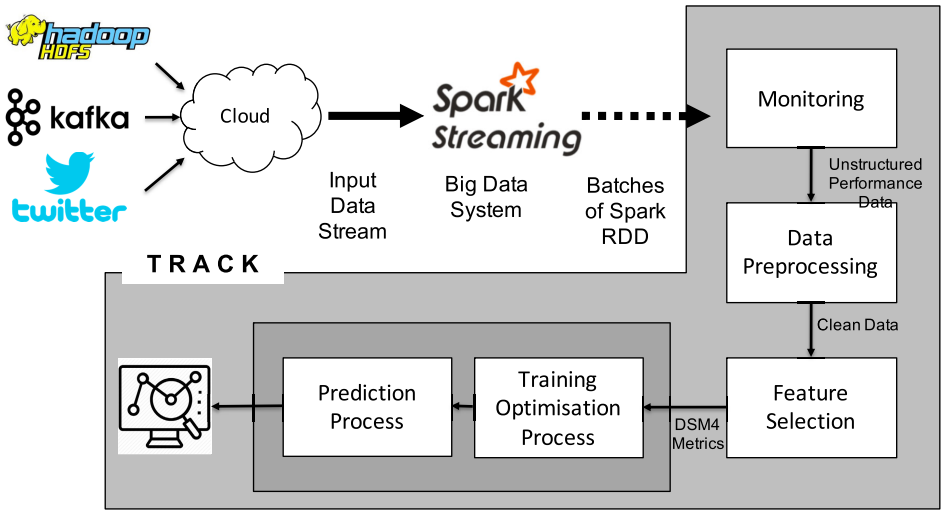


FIGURE 2. TRACK-Plus methodology for anomaly detection.

Equation (1) provides the recursive procedures that compute all error signals δ_h^p for all units in the hidden layers [45]. The F' is the derivative of squashing function for the k^{th} unit in the neural network and is evaluated at the network input (s_h^P) for that unit, where P is the input features vector, N_o number of units in output layer, h is a hidden unit, o is an output unit, and w_{ho} is the weight of the connection between unit in hidden and output layer.

$$\delta_h^p = F'(s_h^P) \sum_{o=1}^{N_o} \delta_o^p w_{ho} \tag{1}$$

The traditional neural networks contain three layers, which input, hidden, and output layer. There are more complex structures of neural networks that require additional execution time and computing resources such as convolutional neural networks, a type of deep neural networks. These neural networks are usually used for image processing. The neural networks model used here has fewer input features (less than 30 features) and output classes than what is used in image processing classification. Therefore, in our case, neural networks with three layers achieve accurate performance classifications.

C. BAYESIAN OPTIMIZATION

The proposed methodology revolves around using Bayesian Optimization (BO) to find the optimal dataset size and configuration parameters for training the neural network to generalize the model so it will detect anomalous behaviors in the Spark Streaming system.

When utilizing BO, there are two main choices to make: using prior over functions and type of acquisition function [46]. It is essential to choose prior over functions to express assumptions about the optimized function. There are different types of acquisition functions, such as *Expected Improvement* [46], *Probability of Improvement* [47], *Lower Confidence Bound* [48], and *Per Second and Plus*. Each type of acquisition function is further discussed in [49].

V. METHODOLOGY

In this section, we introduce TRACK and TRACK-Plus, methodology driven by Bayesian Optimization (BO) and neural networks to train and detect, classify performance anomalies in Apache Spark Streaming systems. Figure 2 shows the TRACK processes of anomaly detection.

A. MACHINE LEARNING MODEL

The neural networks model is used to accurately detect anomalous performance within in-memory Big Data systems such as Apache Spark. The proposed neural networks model in [41] with backpropagation and conjugate gradients is used to train the neural networks to update values of weights and biases in networks. The scaled conjugate is used because it is often faster than other gradient algorithms [50], especially for time-dependent applications such as real-time stream processing.

The neural networks model uses a *Sigmoid* transfer function equation(2) as an activation function, where x includes values of input values to neuron, wights, and bias. *Softmax* transfer function is used in the output layer to handle classification problems with multiple classes of anomalies. For a cost function, *cross-entropy* is used to evaluate the performance of neural networks model. *Cross-entropy* is used because it has significant practical advantages over squared-error cost functions [51].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The proposed neural networks contain three types of layers. The first type of layer is the input layer, which includes a number of neurons equal to the number of input features. The second type of layer is the hidden layer, which has a number of layers (1, 2, or 3) and number of neurons determined using a *trial and error* method, choosing a number between the sizes of input features n_i and output classes n_o [52]. A hidden layer size between n_i and n_o satisfies our goal in achieving accurate results. In our case, the hidden layer size of 5,

10, 15, and 20 achieve 98%, 99%, 96%, and 96% F-scores, respectively. The Hidden layer with ten neurons achieves the highest F-score with the Spark Streaming workload. The output layer contains a number of neurons equal to the number of target classes (types of anomalies), where each neuron generates boolean values: either 0 for normal behavior or 1 for anomalous behavior.

TRACK and TRACK-Plus use Bayesian Optimization to find the optimal training dataset size and configuration parameters to efficiently train the anomaly detection model to achieve high accuracy in a short period of time. Due to its simplicity and tractability, we chose the Gaussian process prior for our proposed model. The acquisition function is used to evaluate a point x based on the posterior distribution function to guide exploration and evaluate the next point [46].

The *Expected Improvement* acquisition function in [53] is used to evaluate the expected performance improvement in the neural networks detection model $f(x)$ and ignore any values that increase the error rate of the model. In other words, x_{best} is the location of the smallest posterior mean (optimal workload configuration) and $\mu_Q(x_{best})$ is the smallest value of the posterior mean. The expected improvement can be described as follows:

$$EI(x) = E_Q[max(0, \mu_Q(x_{best}) - f(x))] \tag{3}$$

E_Q indicates the expectation assumed under the posterior distribution given the evaluations of f at $x_1, x_2, \ldots, x_n$. The time to assess the objective function may vary depending on the region [53].

To improve the performance of the proposed methodology, our TRACK method uses a *customized* acquisition function that utilizes time weighting and the *Expected Improvement* for the acquisition function. The *Expected Improvement* acquisition function assesses the current improvement in the objective function and avoids all outputs that may undermine the performance of objective function output. In addition, the acquisition function operates such that during the evaluation of the objective function by the BO model, another Bayesian model (time-weighting) evaluates the time of the objective function [53]. The final acquisition function is as follows:

$$EI_PS(x) = \frac{EI(x)}{\mu_t(x)} \tag{4}$$

$$EI_PS(x) = \frac{E_Q[max(0, \mu_Q(x_{best}) - f(x))]}{\mu_t(x)} \tag{5}$$

where $\mu_t(x)$ describes the posterior mean of the Gaussian process model timing [53]. A coupled constraint is evaluated only by evaluating the objective function. In our case, the objective function is the performance evaluation of the neural networks model by calculating the F-score. The coupled constraint is that the F-score of the model is not less than a predetermined value (e.g., 90%). The model has several points that are equal to the number of all possible combination parameters of Spark Streaming workload configurations.

Algorithm 1: Training and Testing Methodology for TRACK

Input: Predefined anomaly detection performance $\mathcal{F}$, Workload configuration space $\mathcal{X}$, and system metrics dataset $\mathcal{D}$

Output: Optimal trained neural network model $\mathcal{M}$, which is able to identify anomalies within Spark Streaming with the high predefined F-score in the least amount of time.

Configuring streaming workload benchmark
Workload generation with configuration space $\mathcal{X}$
Streaming workload from network $\mathcal{W} \rightarrow$ Spark system
System profiling to collect performance dataset
Data cleansing and preprocessing $\rightarrow \mathcal{D}$
$DSTrain = 75\%$ of $\mathcal{D} \leftarrow$ total training dataset
$DSTest = 25\%$ of $\mathcal{D} \leftarrow$ total testing dataset
$DSTrain_c$ is empty
$F = 0 \leftarrow$ current f-score
Default_Net_Config: 3 *layers*, 10 units in hidden layer, and *cross-entropy*
$i =$ zero
while $(\,(\,F \leqslant \mathcal{F})\ AND\ (i \leqslant size(\mathcal{X}))\,)$ **do**
  $\mathcal{X}_i = EI_PS(\mathcal{X}) \leftarrow$ acquisition function
  $DSTrain_c = DSTrain_c + DSTrain_{X_i} \leftarrow$
  $\mathcal{M} =$ TrainNN($DSTrain_c$, *NetConfig*) $\leftarrow$ train model on the new dataset configuration
  $\mathcal{F} =$ Max($Fscore(\mathcal{M}(\text{DSTest}))$, $\mathcal{F}$)
  $i = i + 1$

B. MODEL TRAINING, VALIDATION AND TESTING

The Spark Streaming system is randomly injected with anomalies to test the proposed anomaly detection model. For the training process (covers local training, local validation, and local testing), the dataset for every combination of workload configuration parameters (e.g., size s and rate r) is divided into two sets: 75% for model training (DSTrain) and 25% for a global testing dataset (DSTest), as shown in Figure 3. The local DSTrain set for the model is divided into three subsets: local training (70%), local validation (15%), and local testing (15%). The training subset is used to train the model, whereas the validation subset is used to validate the model and to avoid overfitting and underfitting issues. The local testing subset is used to test the model against a single combination of configuration parameters for Spark Streaming workloads. The DSTest set is used to globally test the

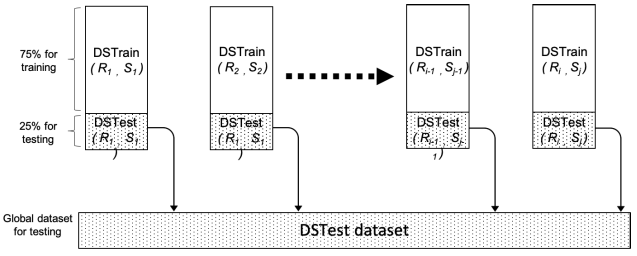


FIGURE 3. Dividing dataset into *DSTrain* and *DSTest* sets for training and testing, respectively.

model, which includes 25% from each possible combination of Spark Streaming workload configuration parameters. This subset is used to independently assess the trained model and to generalize the model.

The streaming workload configurations consist of all the possible combinations of configuration parameters of $Rate_n$ and $Size_m$, for a total of $n \times m$ combinations ($n \times m$ *DSTrain*). The training part of the dataset (*DSTrain*) is divided into 10 equal subsets to find the ideal size of the training dataset. For example, the dataset *DSTrain* workload configuration with rate r_i and size s_j is divided into 10 subsets according to the following equation:

$$DSTrain_{r_i,s_j} = DSTrain_{r_i s_j,1} + \ldots + DSTrain_{r_i s_j,10} \quad (6)$$

The total number of all the possible data subsets is $n \times m \times 10$, so it would be challenging and time-consuming to find the optimal combination of configuration parameters and dataset sizes to train the model. More detailed information about TRACK and TRACK-Plus is presented in Algorithm 1 and Algorithm 2. To assess the proposed model, we use a well-known standard classification performance metric, which is F-score (F), defined in the Appendix alongside the standard metrics of Precision (P) and Recall (R). Further information is provided in the Appendix about Precision, Recall, and F-score.

C. FEATURE SELECTION

The Spark system is monitored at all times and we consider different levels of logging, ranging from Spark jobs measurements to the complete availability of Spark task execution logs, which are used in [41]. These logs provide a reflection of the full details of a Spark system performance. The performance monitoring happens in the background without generating any noticeable overhead in the Spark system.

In this work, we extend the method proposed in [41], called DSM4, which has been built upon the list of Spark performance metrics presented in [41]. DSM4 examines the internal Apache Spark architecture and the Directed Acyclic Graph (DAG) of the Spark application by relying on information from Apache Spark systems. This information includes Spark executors, shuffle read, shuffle write, memory spill, java garbage collection, tasks, stages, jobs, applications, identifications, and execution timestamps for Spark resilient distributed datasets (RDDs). The collected Spark performance metrics are in time series and manually labeled as either normal or anomalous before they are passed as inputs to the proposed model. The proposed methodology assumes that the collected data is pre-processed to ensure the exclusion of any mislabeled training instances and to validate the datasets before passing them to the BO and neural networks model to improve their quality. For example, we avoid duplicated task measurements and exclude samples if features are missing as a result of the monitoring service level anomalies.

VI. EVALUATION

This section evaluates the proposed methodology using a random search (RS) algorithm as a baseline for the same datasets, which are generated from the Apache Spark Streaming system.

Algorithm 2: Training and Testing Methodology for TRACK-Plus

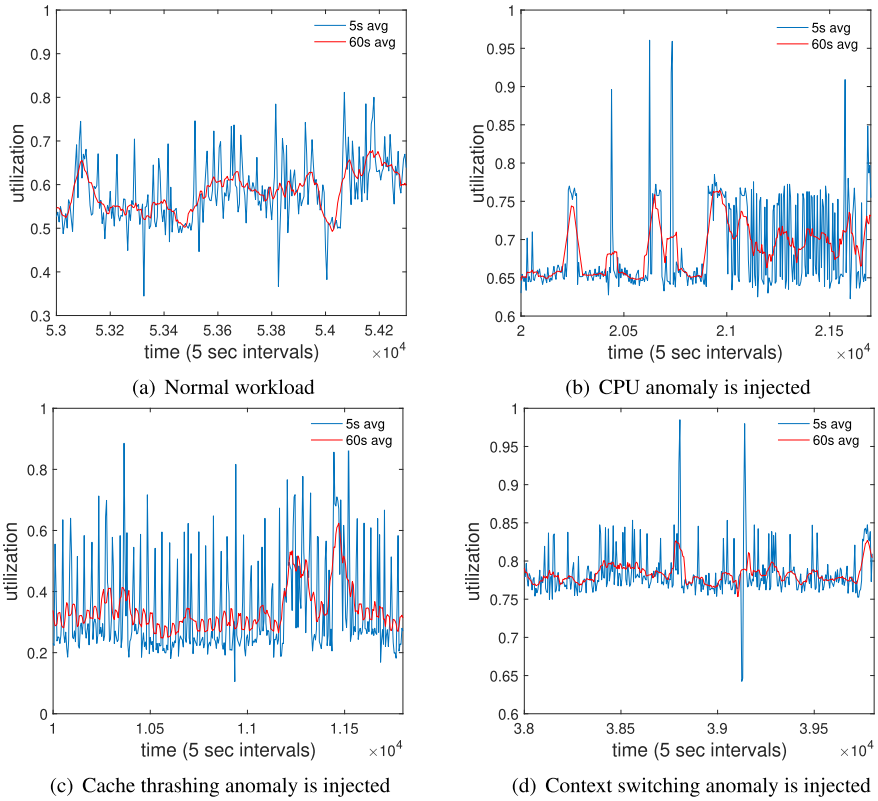
Input: Predefined anomaly detection performance $\mathcal{F}$, Workload configuration space $\mathcal{X}$, and system metrics dataset $\mathcal{D}$

Output: Optimal hypertuned trained neural network model $\mathcal{M}$, which is able to generate an agile model to classify anomalies within Spark Streaming with the high predefined F-score in the least amount of time.

```
Configuring streaming workload benchmark
Workload generation with configuration space 𝒳
Neural Networks with configuration space 𝒩𝒩
Streaming workload from network 𝒲 → Spark system
System profiling to collect performance dataset
Data cleansing and preprocessing → 𝒟
DSTrain = 75% of 𝒟 ← total training dataset
DSTest = 25% of 𝒟 ← total testing dataset
DSTrain_c is empty and F = 0 ← current f-score
Default_Network_Configurations: ℒ layers, 𝒰
  units in hidden layer, and 𝒫 Performance function
i = zero
while ( ( F ⩽ ℱ) AND (i ⩽ size(𝒳)) ) do
    X_i = EIPS(𝒳) ← acquisition function finds next
      workload configurations
    DSTrain_c = DSTrain_c + DSTrain_{X_i} ←
    j = zero and i = i + 1
    while ( ( F ⩽ ℱ) AND (j ⩽ size(𝒩𝒩)) ) do
        NetConfig_j = EIPS(𝒩𝒩) ← acquisition
          function finds next neural networks
          configuration
        ℳ = TrainNN(DSTrain_c, NetConfig_j ) ← train
          neural network model on the new dataset
          configuration
        ℱ = Max( Fscore(ℳ(DSTest)), ℱ )
        j = j + 1
    ℳ = TrainNN(DSTrain_c, NetConfig ) ← train
      neural network model on the new dataset and
      hyperparameters configuration
    ℱ = Max( Fscore(ℳ(DSTest)), ℱ )
```

A. EXPERIMENTAL TESTBED

The experiments are conducted on a Spark Streaming system with 16 core Intel(R) Xeon(R) CPU 2.30 GHz, 32 Gb RAM, Ubuntu 16.04.3, and 2 TB of storage. The Apache Spark is deployed with the Spark Standalone Cluster Manager, 16 executors, and a first-in-first-out scheduler option for deployment. Performance monitoring and data collection are done in the background without causing any noticeable overhead on the system. *Spark History* is used to actively record the performance metrics of internal Spark architecture, such as Spark DAG jobs, stages, and tasks.



(a) Normal workload

(b) CPU anomaly is injected

(c) Cache thrashing anomaly is injected

(d) Context switching anomaly is injected

FIGURE 4. CPU utilization for Spark Streaming workload with normal and anomalous performance.

B. WORKLOAD GENERATION

To evaluate the accuracy of the proposed anomaly detection methodology, we developed the customized *WordCountExp* benchmark for Big Data stream processing systems to generate datasets for training and testing purposes. The workloads are exponentially generated (with exponential distribution) as messages sent through the system network to the data stream processing system with some predefined characteristics such as the rate of sending messages per second and the size of messages. *WordCountExp* is used extensively with many different configurations to evaluate and compare the results of the proposed methodology within in-memory Spark Streaming systems. More than 960 experiments are conducted and 230 Gb of data are collected from the Spark Streaming system, which we use to evaluate the proposed work. The dataset covers four types of injected anomalies within Spark Streaming workloads: normal, CPU anomaly, cache thrashing, and context switching. CPU utilization of the Spark system is shown with different types of anomalous performance in Figure 4.

WordCount is a conventional CPU-intensive benchmark and is widely accepted as a standard micro-benchmark for Big Data platforms [37], [54]–[57]. The WordCount benchmark splits each line of text into multiple words, then aggregates the total number of times each word appears throughout and updates an in-memory map with the words as the key and the frequency of the words as the value. Figure 5 shows a wordcount example of Spark Streaming that receives a streaming workload from a local network to count the number

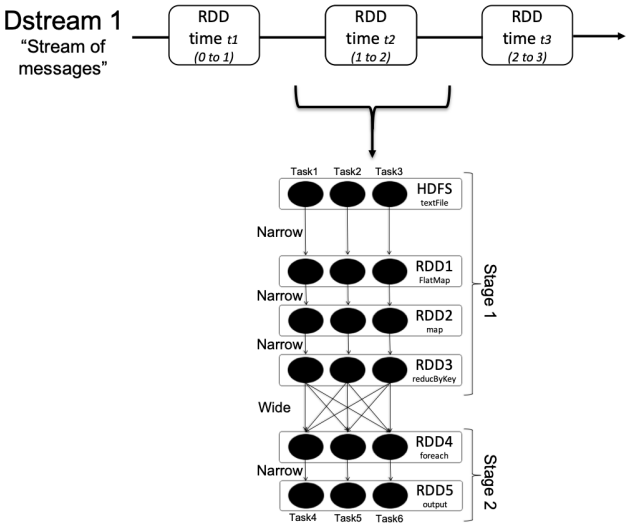


FIGURE 5. Spark Streaming with WordCount example of DStream.

of words per message. The Main *DStream* data is divided into many RDDs for a certain time interval, then some Spark operations, such as wide and narrow operations, are done to count the number of words in each Spark RDD. More details about Spark operations are discussed in [41].

C. ANOMALY INJECTION

To inject different types of anomalies, the open-source tool (*stress-ng*) is used to evaluate the proposed methodology with the Spark Streaming system [41]. A list of performance

TABLE 2. Types of anomalies.

Type #	Description
CPU	Spawn n workers running the *sqrt()* function
Memory	Continuously writing to allocated memory in order to cause memory stress.
Cache thrashing	n processes perform random widespread memory read and writes to thrash the CPU cache.
Context switching	n processes force context switching.

anomalies is used to generate CPU stress, cache thrashing stress, and context switching stress as shown in Table 2. The CPU stress spawns n workers to run the sqrt() function; the cache thrashing stress causes n processes to perform random widespread memory read-and-writes to thrash the CPU cache; and the context switching stress has n processes that forces context switching. The injected anomaly and the used benchmark are configured depending on the objective of the experiment, which will be discussed in Section VII.

VII. RESULTS

The proposed methodology is evaluated on an isolated Spark Streaming system, discussed in Section VI-A. We avoid using a virtual Spark System, which ensures that all performance metrics are accurately measured.

A. FINDING THE IDEAL WORKLOAD CONFIGURATION FOR MODEL TRAINING

The previous discussion regarding the motivating example (Section III) describes the need to find the ideal single workload configurations set (e.g., rate r_i and size s_j) that could be used to train the proposed anomaly detection model to pinpoint the abnormal behavior with the highest possible F-score. This facilitates the use of a single workload configuration to be generalized and used to detect anomalies with the other workload configurations. The Spark Streaming workload has all possible combinations of *rates 1, 8, 16, and 32 message/sec* and *sizes 1, 10, 100, and 1000 line/message*, for a total of 16 combinations.

A Bayesian Optimization (BO) and neural networks model (described in Section IV-C and in IV-B) are used to address the need for determining the ideal single workload configuration (rate r_i and size s_j) with the minimum number of running experiments n. To ensure accurate results, the experiments are conducted 50 times, then the average of n is calculated. The results show that the ideal F-score is reached with the minimum number of running experiments (n=*8*), which is 50% less than the total number of possible configurations (n=*16*).

Figure 6 shows the performance results of the proposed model when it is individually trained on each workload configuration (rate r_i and size s_j) and tested against all possible combinations of streaming workload configurations using BO and neural networks. The estimated objective value is the deviation from the ideal F-score (*error = 1- F-score*).

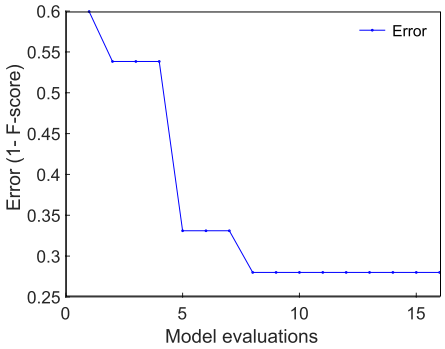


FIGURE 6. Training process results for each workload configuration, tested on all possible combinations of streaming workload configurations using Bayesian Optimization and neural networks.

Figure 6 illustrates that with the given dataset, the workload configuration ($r = 32$, $s = 1$) can be used to train the anomaly detection model to detect abnormal behavior with all other streaming workload configurations with the highest F-scores equaling 72% after running only 8 of 16 experiments. The next section explores a new approach to optimize the model and obtain a higher F-score using less time in training processes.

B. BAYESIAN OPTIMIZATION MODEL TO TRAIN ANOMALY DETECTION TECHNIQUE

A BO model (discussed in Section IV-C) is used to find the optimal size of the training dataset and the streaming workload configurations set to achieve the highest accuracy with the least time spent training the proposed anomaly detection model. The model training and datasets of anomaly detection are comprehensively discussed in Section V-B and V-C. Figure 7 depicts a comparison of BO and RS to reach a predefined F-score with the fewest training steps from the total 160 steps. The conducted experiments have workloads containing both normal and anomalous CPU behaviors with all possible combinations of workload configurations. Figure 7 shows the average of the 50 experiments where the neural networks model is trained using BO to achieve the predefined F-score. With BO, the trained model reaches a 95% F-score in 21 steps, whereas an RS uses 28 steps (enhanced by 25%). This proves that the proposed model can reduce the time and computation process by 25%. Table 3 shows the performance of five different types of acquisition functions that are used with BOs.

Two other types of anomalies may disrupt the performance of the Big Data stream processing system. These two types

TABLE 3. Performance of different types of acquisition functions to reach 95% F-score. The right column shows the average number of steps for 50 experiments to find the ideal dataset size to train the model.

Acquisition Functions	Number of steps
BO expected-improvement	64.3
BO expected-improvement-plus	21
BO expected-improvement-per-second	30.2
BO lower-confidence-bound	54.9
BO probability-of-improvement	43.4

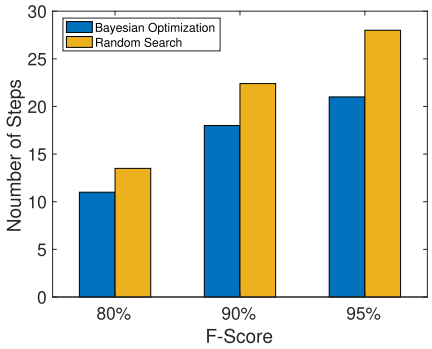


FIGURE 7. Comparison of Bayesian Optimization (BO) and RS for reaching a predefined F-score with the fewest training steps. Workloads have all possible combinations of parameters and CPU anomalies.

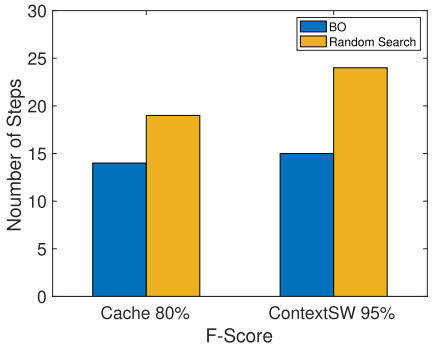


FIGURE 8. Comparison of BO and RS to reach predefined F-scores (80% for cache stress and 90% for context switching stress) with the minimum number of steps. Workloads have all possible combinations of parameters.

are cache thrashing and context switching. The proposed model can detect both cache thrashing and context switching anomalies with F-scores of 80% and 95%, respectively. Figure 8 shows that the proposed model outperforms RS by more than 25% and can reduce the amount of computations from 160 experiments to 14, as can be seen in Figure 8 with cache thrashing anomalies.

C. SENSITIVITY ANALYSES OF TRAINING DATASET SIZE

In this subsection, the impact of the training dataset size is examined to prove the robustness of the proposed model. The amount of anomalous Spark tasks decrease by 50% to 75% of the anomalous workload in Section VII-B. Table 4 depicts the impact of the Spark workload training set size on the proposed stream anomaly detection model. The BO with neural networks model achieves the highest performance in detecting all three types of performance anomalies in Spark

TABLE 4. Sensitivity analysis demonstrating the impact of reducing the overall anomalous training dataset size by 50% to 75%. BO is compared against RS to assess when each would reach ideal performance (95% F-score) with the fewest possible steps and the least training data. Workloads contains all possible combinations of rates 1, 8, 16, and 32 *message/sec* and sizes 1, 10, 100, and 1000 *line/message*.

Algorithm	CPU	Cache	Context Sw
BO	38	19.9	16
Random Search	44	24	25

TABLE 5. Testing the proposed model against new unseen workload configurations with three types of performance anomalies.

Types of Anomalies	F-score $\pm$ Standard deviation
CPU	0.93$\pm$0.01
Cache Thrashing	0.77$\pm$0.02
Context Switching	0.72$\pm$0.04

Streaming systems. This proves that the proposed model is robust against changes in the size of the overall input training datasets.

D. NEW UNSEEN WORKLOAD CONFIGURATIONS

This section presents the training of the proposed model with predefined workload configurations (*rates 1, 8, 16, and 32 message/sec* and *sizes 1, 10, 100, and 1000 line/message*) and generalizes the model to perform just as accurately with new unseen workload configurations (e.g., $r_i = 20$ and $s_j = 150$). In this case, the workload is more realistic and reflects the workload characteristics of the real stream processing system in the production environment.

For the training phase, the same BO and neural network configurations in Section VII-B are used to train the model on predefined workload configurations (*rates 1, 8, 16, and 32 message/sec* and *sizes 1, 10, 100, and 1000 line/message*) to reach a 95% F-score for detecting CPU performance anomalies. For the testing phase, the final model of the training phase is used to detect anomalous behavior but with new unseen workload configurations. The rate could be between 1 to 32 and the size could have ranged from 1 to 1000. The total number of possible configuration combinations is 32k.

Table 5 shows the performance of the proposed model when it is tested against three types of anomalies (i.e., CPU, cache thrashing, and context switching). As seen in Table 5, the proposed anomaly detection model can be trained on 16 workload configurations to be generalized to detect anomalies against 32k different workload configurations.

E. DETECTING AND CLASSIFYING PERFORMANCE ANOMALIES

In this section, we show that TRACK will not only detect anomalous performance but also classify workloads into four types: normal, CPU anomaly, cache anomaly, and context switching anomaly. The anomaly detection using TRACK achieves 74% for detecting and classifying Spark Streaming performance, as seen in Table 6. The next section introduces a new optimized version of TRACK called TRACK-Plus to find the ideal neural network configuration to accelerate the search process and improve anomaly classification.

F. TRACK-PLUS FOR OPTIMIZING THE CHOICE OF NEURAL NETWORKS ARCHITECTURE

The performance of TRACK-Plus is evaluated using the two BO models discussed in Section IV-C. The first BO1 is used to find the ideal dataset training size as described

TABLE 6. Performance of TRACK for detecting and classifying anomalies based on their root causes.

Type of workload	Recall	Pres	F-score
Normal	0.89	0.91	0.90
CPU	0.56	0.55	0.55
Cache Thrashing	0.97	0.95	0.96
Context Switching	0.54	0.56	0.55
Average	0.74	0.74	0.74

TABLE 7. All possible optimized configurations for TRACK-Plus including two BO models.

BO#	Parameters	Possible Configurations
BO1	Message Rate (*message/sec*)	1, 8, 16, or 32
	Message Size (*line/message*)	1, 10, 100, or 1000
	Training Dataset Size	1, 2, 3, 4, 5, 6, 7, 8, 9, or 10
BO2	Number of Neurons	5, 10, 15, or 20
	Number of Hidden Layers	1, 2, or 3
	Performance Function	mae, mse, sae, sse, or crossentropy

in Section V-B. BO1 optimizes the choices for three Spark Streaming workload configurations, which are the rate of messages per second (1, 8, 16, and 32), message size (1, 10, 100, and 1000), and the size of the training dataset (1 to 10). The total number of possible configurations is $4 \times 4 \times 10$, which comes close to 160 different possible combinations.

The objective of the second BO2 is to automate the search to achieve the most efficient architecture of neural networks (with a predefined list of configurations) by optimizing the tuning process of the hyperparameters of the neural networks. In practice, different configurations of hyperparameters can significantly impact the performance of the neural networks. In this study, we focus more on hyperparameters related to neural network training and structure, including the number of hidden layers, number of neurons in each layer, and performance functions. Five well-known performance functions have been examined in TRACK-Plus, which are mean absolute error, mean squared error, sum absolute error, sum squared error, and cross-entropy. The total number of possible configuration combinations for BO2 is $5 \times 3 \times 4 = 60$ different possible configurations. Details of the configuration parameters of the two BO models can be found in Table 7.

Even with the limited number of configurations to train the anomaly detection technique, TRACK-Plus offers an efficient solution in finding the ideal training dataset size and the most efficient neural network configurations to accurately detect the anomalous performance within the Spark Streaming system. For example, Table 7, with the list of the total number of possible configuration combinations, shows that there are $160 \times 60 = 9600$ possible configurations. It is clear that finding the ideal configurations with which to train the anomaly detection model is more time-consuming and resource intensive when using either the traditional search or the manual configurations.

TABLE 8. The ideal configuration for the neural networks in terms of performance function, number of layers, and number of neurons in each layer.

Parameters	Configurations	Number of Selection
Performance Func	mae	16
	mse	3
	sae	21
	sse	8
	crossentropy	2
#Neurons/Layer	5	18
	10	14
	15	8
	20	10
#Layers	1	25
	2	9
	3	16

Table 8 shows the average results of 50 experiments where the TRACK-Plus optimizes the training process of anomaly detection to achieve the predefined F-score, which is 70% (the highest possible F-score for classifying the anomalies). With the given conditions of Spark Streaming workloads, we find that the ideal neural networks configurations are *sae* performance function, five neurons/layer, and one hidden layer.

VIII. CONCLUSION

To develop effective fault-tolerant system performance, it is vital to detect anomalous performance and service level disruption events within data intensive systems. The growing complexity of Big Data systems makes performance anomaly detection more challenging, especially for critical streaming workload applications in distributed systems environments. Therefore, the performance of in-memory processing technology like Apache Spark Streaming must be thoroughly investigated to pinpoint the causes of performance anomalies.

Collecting all possible performance measurements from Big Data systems to train the anomaly detection system is computationally expensive, especially for critical systems such as online banking, stock trading, and air traffic control systems. Even with the WordCount Spark Streaming application (only has two parameters r and s), it is considered a time-consuming and costly intensive computing to find the ideal dataset size to efficiently train the anomaly detection model so it will comprehensively cover all seen and unseen anomalies.

This paper contributes by addressing the challenge of anomalous identification by proposing a new hybrid learning solutions, TRACK and TRACK-Plus, for anomaly detection within in-memory Big Data systems. The anomaly detection and tuning method are developed using Bayesian Optimization and neural networks to train the model with a limited budget and limited computing resources. As can be seen from the experimental results, the proposed model efficiently finds the optimal training dataset size and configuration parameters to accurately identify different types of performance anomalies in Big Data systems. The proposed model achieves the

highest accuracy (95% F-score) in significantly less time (80% less than normal). A validation based on a real dataset for the Apache Spark Streaming system has been provided to demonstrate that the proposed methodology identifies the performance anomalies, the ideal configuration parameters, and the training dataset size with up to 75% fewer experiments. Finally, the proposed solutions not only identifies anomalous performance with a high F-score but also classifies anomalies, thereby saving considerable time in training the model. In addition, the proposed model can be easily generalized to cover unforeseen workload configurations.

In terms of future work, it is crucial to deeply investigate an anomaly detection and prediction for systems that contain both batch and stream processing workloads at the same time. Such systems will have more increasing complexity and performance fluctuation, which may need more effective anomaly detection solutions. Exploring deep Learning algorithms may hold opportunities to accurately detect and predict the performance anomaly in distributed complex systems.

APPENDIXES

Recall (also called *Sensitivity*) and *Precision* performance measures are used in this paper to evaluate Track and Track-Plus for anomaly detection classifiers. These performance metrics are commonly used and standard metrics for quantifying the accuracy of the classifiers [58]. The following are the anomaly classification classes and their notations:

- True positive (*tp*): a correct detection of anomalies
- False positive (*fp*): a detection of anomalies when it does not exist
- False negative (*fn*): a missed detection of anomalies when it exists.

$$R = \frac{tp}{tp + fn} \tag{7}$$

R is the Recall and it answers this question: of all the samples which are anomalies, how many of those are correctly detected?. *Recall* assesses the effectiveness of a classifier in identifying positive samples. *Recall* will become higher when the anomaly detection classifier can detect all anomalies.

$$P = \frac{tp}{tp + fp} \tag{8}$$

P is *Precision* and it answers this question: how many of those samples which are labeled as anomaly are actually anomaly?. *Precision* quantifies class agreement on how many samples classified as positive (*anomaly*) are, indeed, positive. It assesses the reliability of the detection method when it detects anomalies.

$$F = 2\frac{PR}{P + R} \tag{9}$$

F-score reflects the relations between data's anomalous labels and those given by a classifier. *F-score* captures the trade-off between *Recall* and *Precision*. It shows a summary score computed for harmonic mean of *Recall* and *Precision*. Throughout this paper, we use the F-score as the main performance metric. The formulas of *Recall*, *Precision*, and *F-score* reflect the quality of classifier in detecting the positive samples (anomalies in our case), without paying significance attention on the correct classification [59].

REFERENCES

[1] S. Fu, J. Liu, and H. Pannu, "A hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines," in *Proc. Int. Conf. Adv. Data Mining Appl.* Nanjing, China: Springer, 2012, pp. 726–738.

[2] X. Gu and H. Wang, "Online anomaly prediction for robust cluster systems," in *Proc. IEEE 25th Int. Conf. Data Eng.*, Mar. 2009, pp. 1000–1011.

[3] S. Baek, D. Kwon, J. Kim, S. C. Suh, H. Kim, and I. Kim, "Unsupervised labeling for supervised anomaly detection in enterprise and cloud networks," in *Proc. IEEE 4th Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)*, Jun. 2017, pp. 205–210.

[4] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *Proc. IEEE 16th Intl Conf Dependable, Autonomic Secure Comput.*, Aug. 2018, pp. 151–158.

[5] E. W. Fulp, G. A. Fink, and J. N. Haack, "Predicting computer system failures using support vector machines," in *Proc. WASL*, vol. 8, Dec. 2008, p. 5.

[6] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerg. Artif. Intell. Appl. Comput. Eng.*, vol. 160, no. 1, pp. 3–24, 2007.

[7] S. Fu, "Performance metric selection for autonomic anomaly detection on cloud computing systems," in *Proc. IEEE Global Telecommun. Conf.*, Dec. 2011, pp. 1–5.

[8] W. Qi, Y. Li, H. Zhou, W. Li, and H. Yang, "Data mining based root-cause analysis of performance bottleneck for big data workload," in *Proc. IEEE 19th Int. Conf. High Perform. Comput. Commun.*, Dec. 2017, pp. 254–261.

[9] S. Lu, X. Wei, B. Rao, B. Tak, L. Wang, and L. Wang, "LADRA: Log-based abnormal task detection and root-cause analysis in big data processing with spark," *Future Gener. Comput. Syst.*, vol. 95, pp. 392–403, Jun. 2019.

[10] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based abnormal task detection and root cause analysis for spark," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 389–396.

[11] H. Herodotou, Y. Chen, and J. Lu, "A survey on automatic parameter tuning for big data processing systems," *ACM Comput. Surveys*, vol. 53, no. 2, pp. 1–37, Jul. 2020.

[12] A. S. Alnafessah and G. Casale, "TRACK: Optimizing artificial neural networks for anomaly detection in spark streaming systems," in *Proc. 13th EAI Int. Conf. Perform. Eval. Methodol. Tools*, May 2020, pp. 188–191.

[13] G. Ananthanarayanan, "Reining in the outliers in map-reduce clusters using mantri," in *Proc. OSDI*, Oct. 2010, vol. 10, no. 1, p. 24.

[14] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 1–35, Sep. 2015.

[15] P. Garraghan, X. Ouyang, R. Yang, D. McKee, and J. Xu, "Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters," *IEEE Trans. Services Comput.*, vol. 12, no. 1, pp. 91–104, Jan. 2019.

[16] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, Oct. 2016.

[17] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, Oct. 2004.

[18] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.

[19] M. Lin, Z. Yao, F. Gao, and Y. Li, "Toward anomaly detection in iaas cloud computing platforms," *Int. J. Secur. Its Appl.*, vol. 9, no. 12, pp. 175–188, 2015.

[20] T. Huang, Y. Zhu, Q. Zhang, Y. Zhu, D. Wang, M. Qiu, and L. Liu, "An LOF-based adaptive anomaly detection scheme for cloud computing," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2013, pp. 206–211.

[21] H. Zhang, J. Rhee, N. Arora, S. Gamage, G. Jiang, K. Yoshihira, and D. Xu, "CLUE: System trace analytics for cloud service performance diagnosis," in *Proc. IEEE Netw. Operations Manage. Symp. (NOMS)*, May 2014, pp. 1–9.

[22] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection," *Inf. Sci.*, vol. 177, no. 18, pp. 3799–3821, Sep. 2007.
[23] Y. Si, Y. Wang, and D. Zhou, "Key-Performance-Indicator-Related process monitoring based on improved kernel partial least squares," *IEEE Trans. Ind. Electron.*, early access, Feb. 13, 2020, doi: 10.1109/TIE.2020.2972472.
[24] Y. Wang, Y. Si, B. Huang, and Z. Lou, "Survey on the theoretical research and engineering applications of multivariate statistics process monitoring algorithms: 2008-2017," *Can. J. Chem. Eng.*, vol. 96, no. 10, pp. 2073–2085, Oct. 2018.
[25] J. P. Magalhaes and L. M. Silva, "Detection of performance anomalies in Web-based applications," in *Proc. 9th IEEE Int. Symp. Netw. Comput. Appl.*, Jul. 2010, pp. 60–67.
[26] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni, "R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads," in *Proc. Int. Conf. Distrib. Syst. Platforms Open Distrib. Process.*, 2007, pp. 244–265.
[27] D. A. Menasce, "TPC-W: A benchmark for e-commerce," *IEEE Internet Comput.*, vol. 6, no. 3, pp. 83–87, May 2002.
[28] T. Kelly, "Detecting performance anomalies in global applications," in *Proc. WORLDS*, vol. 5, 2005, pp. 42–47.
[29] L. Yang, C. Liu, J. M. Schopf, and I. Foster, "Anomaly detection and diagnosis in grid environments," in *Proc. ACM/IEEE Conf. Supercomput.*, 2007, pp. 1–9.
[30] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "SparkBench: A comprehensive benchmarking suite for in memory data analytic platform spark," in *Proc. 12th ACM Int. Conf. Comput. Frontiers*, 2015, p. 53.
[31] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 111–124.
[32] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic Internet services," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2002, pp. 595–604.
[33] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," *ACM SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 105–118, Oct. 2005.
[34] H. S. Pannu, J. Liu, and S. Fu, "A self-evolving anomaly detection framework for developing highly dependable utility clouds," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2012, pp. 1605–1610.
[35] R. Ren, S. Tian, and L. Wang, "Online anomaly detection framework for spark systems via stage-task behavior modeling," in *Proc. 15th ACM Int. Conf. Comput. Frontiers*, May 2018, pp. 256–259.
[36] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "BigDataBench: A big data benchmark suite from Internet services," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014, pp. 488–499.
[37] N. J. Yadwadkar and W. Choi, "Proactive straggler avoidance using machine learning," Univ. Berkeley, Berkeley, CA, USA, White Paper, 2012.
[38] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *Proc. 38nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 1511–1516.
[39] KDD-99. (1999). *Spark Streaming Programming Guide*. Accessed: May 5, 2020. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[40] M. Lin, S. Chen, G. Chang, L. Liu, and X. Gong, "Research on pca-based anomaly detection system in high-speed network," *J. Comput. Inf. Syst.*, vol. 7, no. 7, pp. 2315–2321, 2011.
[41] A. Alnafessah and G. Casale, "Artificial neural networks based techniques for anomaly detection in apache spark," *Cluster Comput.*, vol. 23, pp. 1–16, Oct. 2019.
[42] Y. Owechko and S. Shams, "Comparison of neural network and genetic algorithms for a resource allocation problem," in *Proc. IEEE Int. Conf. Neural Netw. (ICNN)*, 1994, pp. 4655–4660.
[43] ApacheSpark. (2019). *Spark Streaming Programming Guide*. Accessed: Oct. 1, 2019. [Online]. Available: https://spark.apache.org/docs/latest/streaming-programming-guide.html
[44] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/
[45] B. Kröse, B. Krose, P. van der Smagt, and P. Smagt, *An Introduction to Neural Networks*. Citeseer, 1993.
[46] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2951–2959.
[47] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *J. Basic Eng.*, vol. 86, no. 1, pp. 97–106, Mar. 1964.
[48] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," 2009, *arXiv:0912.3995*. [Online]. Available: http://arxiv.org/abs/0912.3995
[49] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.
[50] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Netw.*, vol. 6, no. 4, pp. 525–533, Jan. 1993.
[51] D. M. Kline and V. L. Berardi, "Revisiting squared-error and cross-entropy functions for training neural network classifiers," *Neural Comput. Appl.*, vol. 14, no. 4, pp. 310–318, Dec. 2005.
[52] K. G. Sheela and S. N. Deepa, "Review on methods to fix number of hidden neurons in neural networks," *Math. Problems Eng.*, vol. 2013, Jun. 2013, Art. no. 425740.
[53] Mathworks. (2019). *Bayesian Optimization Algorithm*. Accessed: Oct. 1, 2019. [Online]. Available: https://uk.mathworks.com/help/stats/bayesian-optimization-algorithm.html#bva8tie-1
[54] S. Qian, G. Wu, J. Huang, and T. Das, "Benchmarking modern distributed streaming platforms," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Mar. 2016, pp. 592–598.
[55] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Proc. IEEE 26th Int. Conf. Data Eng. Workshops (ICDEW)*, Mar. 2010, pp. 41–51.
[56] R. Lu, G. Wu, B. Xie, and J. Hu, "Stream bench: Towards benchmarking modern distributed stream computing frameworks," in *Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput.*, Dec. 2014, pp. 69–78.
[57] R. Han, L. Kurian John, and J. Zhan, "Benchmarking big data systems: A review," *IEEE Trans. Services Comput.*, vol. 11, no. 3, pp. 580–597, Jun. 2018.
[58] G. Casale, C. Ragusa, and P. Parpas, "A feasibility study of host-level contention detection by guest virtual machines," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2013, pp. 152–157.
[59] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manage.*, vol. 45, no. 4, pp. 427–437, Jul. 2009.

AHMAD ALNAFESSAH (Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Computing, Imperial College London. He was a Senior Academic Researcher with the National Centre for AI and Big Data Technologies, KACST, from 2012 to 2017. His research interests include performance engineering for big data systems and AI, with a specific focus on in-memory platforms, the IoT, HPC, complex distributed systems, and cloud computing.

GIULIANO CASALE (Member, IEEE) was a Scientist with SAP Research, U.K., and a Consultant with the Capacity Planning Industry. He joined the Department of Computing, Imperial College London, in 2010, where he is currently a Senior Lecturer in modeling and simulation. He also teaches and does research. He has published more than 100 refereed articles. His research interests include performance engineering, cloud computing, and big data. He has served on the technical program committee of over 80 conferences and workshops. His research was a recipient of multiple awards, such as the Best Paper Award from ACM SIGMETRICS, in 2017. He served as the Co-Chair for several conferences in performance engineering, such as ACM SIGMETRICS/performance. He also serves as the Chair for ACM SIGMETRICS. He serves on the editorial boards of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and *TOMPECS* (ACM).